Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №4

на тему

**ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ(ПОТОКОВ): ВЗАИМНОЕ ИСКЛЮЧЕНИЕ И СИНХРОНИЗАЦИЯ.**

Выполнил: студент гр.253505 Снежко М.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

[1 Формулировка задачи 3](#_Toc179742261)

[2 Краткие теоритические сведения 4](#_Toc179742262)

[3 Описание функций программы 5](#_Toc179742263)

[Заключение 6](#_Toc179742264)

[Список использованных источников 7](#_Toc179742265)

[Приложение А (обязательное) 8](#_Toc179742266)

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Цель лабораторной работы – изучение методов и инструментов организации взаимодействия между процессами (потоками) на основе модели «писатели-читатели», включая использование различных системных объектов и функций для синхронизации доступа к общему ресурсу. В процессе выполнения лабораторной работы также будут рассмотрены типичные проблемы, возникающие при организации многопоточной обработки данных, такие как предотвращение взаимных блокировок, минимизация задержек и обеспечение корректности работы потоков.

В качестве задачи необходимо разработать программу на языке *C++*, моделирующую взаимодействие между писателями и читателями, которые обращаются к общему ресурсу. Модель включает несколько потоков, где писатели генерируют и записывают данные, а читатели считывают их. Каждый поток характеризуется параметрами запроса, моментом обращения и длительностью использования ресурса. В рамках работы обеспечивается корректная синхронизация потоков: исключается одновременное выполнение конфликтующих операций, минимизируются задержки за счет оптимального использования системных объектов, таких как мьютексы, семафоры и события.

Для выполнения задачи потребуется настроить многопоточность с использованием системных средств синхронизации. Писатели и читатели взаимодействуют через общий ресурс, при этом корректность работы модели достигается за счет ограничения одновременного доступа и предотвращения "грязного" чтения данных. В процессе выполнения программы будут исследованы различные параметры, включая интенсивность запросов потоков, длительность работы с ресурсом, количество участников, а также штрафы за блокировку. Статистика, собранная во время работы программы, позволит оценить эффективность модели по таким критериям, как соотношение успешных и неуспешных операций, время активности потоков и пропускная способность системы.

Таким образом, данная лабораторная работа направлена на углубление понимания принципов многопоточной обработки данных и механизмов синхронизации потоков. Реализация модели "писатели-читатели" с возможностью параметризации и анализа результатов позволяет изучить влияние различных факторов на производительность системы. Полученные результаты помогут понять, как изменяемые параметры, такие как количество потоков, интенсивность обращений и длительность обработки, влияют на общую эффективность работы и стабильность системы.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Объект синхронизации – это объект, дескриптор которого можно указать в одной из функций ожидания для координации выполнения нескольких потоков. Несколько процессов могут иметь дескриптор одного и того же объекта синхронизации, что делает возможной синхронизацию между процессами [1].

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений [2].

Операционная система Windows предоставляет механизмы для упрощения обмена данными и обмена данными между приложениями. В совокупности действия, включенные этими механизмами, называются межпроцессными коммуникациями (IPC). Некоторые формы IPC упрощают разделение труда между несколькими специализированными процессами. Другие формы IPC упрощают разделение труда между компьютерами в сети.

Как правило, приложения могут использовать IPC, классифицированные как клиенты или серверы. Клиент – это приложение или процесс, который запрашивает службу от другого приложения или процесса. Сервер – это приложение или процесс, который отвечает на запрос клиента. Многие приложения действуют как клиент, так и сервер в зависимости от ситуации. Например, приложение обработки слов может выступать в качестве клиента при запросе сводной таблицы производственных затрат от приложения электронной таблицы, выступающего в качестве сервера. В свою очередь, приложение электронной таблицы может выступать в качестве клиента при запросе последних уровней инвентаризации из приложения автоматического управления инвентаризацией [3].

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Разработанная программа представляет собой многопоточное приложение, реализующее шаблон взаимодействия "писатели-читатели". Она предназначена для моделирования работы потоков, конкурирующих за доступ к общему ресурсу. Потоки-читатели имитируют операции чтения ресурса, проверяя его доступность. Если ресурс занят писателем, они блокируются до освобождения. Фиксируется время ожидания, и если оно превышает установленный порог, событие регистрируется как заблокированное чтение. Потоки-писатели выполняют запись, получая эксклюзивный доступ к ресурсу. Для этого они дожидаются завершения всех операций чтения. Продолжительность их ожидания также фиксируется, и при превышении порога регистрируется заблокированная запись.

Синхронизация потоков обеспечивается с помощью семафора для управления доступом к ресурсу, мьютекса для изменения глобальных переменных, таких как количество активных читателей и писателей, а также событий для координации операций потоков. В программе реализован сбор статистики, где подсчитывается количество успешных и заблокированных операций чтения и записи. Каждые пять секунд статистика выводится на экран, что позволяет анализировать эффективность взаимодействия потоков. Результат работы программы представлен на рисунке 3.1.

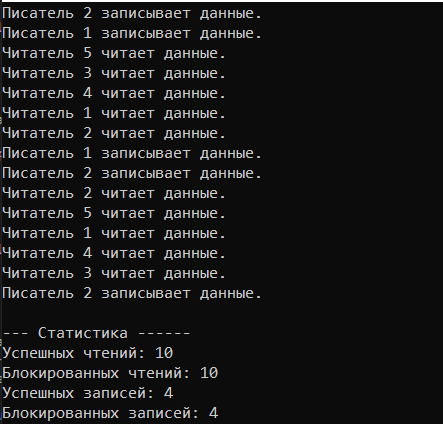


Рисунок 3.1 – Результат работы программы

Таким образом, данная программа демонстрирует работу с многопоточностью и основы конкурентного доступа к ресурсам. Она позволяет изучить влияние параметров, таких как количество потоков, задержка и время выполнения операций, на производительность. Итогом работы является анализ собранной статистики, что подтверждает корректность реализации механизма взаимодействия.

# ЗАКЛЮЧЕНИЕ

Таким образом, в рамках данной лабораторной работы была разработана программа для моделирования взаимодействия потоков на основе модели «писатели-читатели». Основная цель работы заключалась в изучении методов и инструментов синхронизации потоков, необходимых для организации корректного и эффективного совместного использования общего ресурса. Реализация модели позволила изучить принципы обеспечения согласованного доступа, минимизации блокировок и предотвращения конфликтных ситуаций, таких как «грязное» чтение данных.

Разработанная программа представляет собой программную модель, в которой несколько потоков, играющих роли писателей и читателей, взаимодействуют через общий ресурс. Потоки-писатели генерируют данные и записывают их, тогда как потоки-читатели извлекают и обрабатывают эти данные. Для обеспечения корректности взаимодействия использовались системные объекты синхронизации, такие как мьютексы, семафоры и события, которые предотвращают одновременное выполнение несовместимых операций.

Во время выполнения программы исследовались различные параметры модели, включая интенсивность запросов, количество потоков, длительность их активности и блокировок, а также общая эффективность системы. Параметризация модели позволила провести анализ влияния этих факторов на производительность и стабильность работы. В ходе работы также были выявлены и изучены типичные проблемы многопоточной обработки, такие как конкуренция за ресурсы, влияние длительности блокировок на производительность и необходимость балансировки нагрузки между потоками.

Таким образом, лабораторная работа не только способствовала углублению знаний о принципах многопоточной обработки и синхронизации, но и позволила на практике изучить влияние различных параметров модели на общую эффективность системы, предоставив ценный опыт в проектировании многопоточных приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Microsoft "Synchronization objects" – Электронный ресурс. Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/synchronization-objects.

[2] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.

[3] Microsoft "Взаимодействие процессов" – Электронный ресурс. Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/synchronization-objects.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <windows.h>

#include <iostream>

#include <vector>

#include <thread>

#include <random>

#include <chrono>

#include <atomic>

int num\_readers = 5;

int num\_writers = 2;

int min\_read\_time = 700;

int max\_read\_time = 900;

int min\_write\_time = 500;

int max\_write\_time = 800;

int min\_delay\_time = 200;

int max\_delay\_time = 500;

double meaningful\_block\_time = 0.2;

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> read\_dist(min\_read\_time, max\_read\_time);

std::uniform\_int\_distribution<> write\_dist(min\_write\_time, max\_write\_time);

std::uniform\_int\_distribution<> delay\_dist(min\_delay\_time, max\_delay\_time);

int readers\_count = 0;

int writers\_count = 0;

int readers\_count\_waiting = 0;

HANDLE resource\_semaphore;

HANDLE counter\_mutex;

HANDLE reader\_cv, writer\_cv;

std::atomic<int> successful\_reads(0);

std::atomic<int> successful\_writes(0);

std::atomic<int> blocked\_reads(0);

std::atomic<int> blocked\_writes(0);

void reader(int id)

{

while (true)

{

std::this\_thread::sleep\_for(std::chrono::milliseconds(delay\_dist(gen)));

auto start\_time = std::chrono::high\_resolution\_clock::now();

WaitForSingleObject(counter\_mutex, INFINITE);

++readers\_count\_waiting;

if (writers\_count > 0)

{

ReleaseMutex(counter\_mutex);

ResetEvent(reader\_cv);

WaitForSingleObject(reader\_cv, INFINITE);

}

else

{

ReleaseMutex(counter\_mutex);

}

WaitForSingleObject(counter\_mutex, INFINITE);

if (++readers\_count == 1)

{

ReleaseMutex(counter\_mutex);

WaitForSingleObject(resource\_semaphore, INFINITE);

}

else

{

ReleaseMutex(counter\_mutex);

}

WaitForSingleObject(counter\_mutex, INFINITE);

--readers\_count\_waiting;

ReleaseMutex(counter\_mutex);

auto end\_time = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> block\_duration = end\_time - start\_time;

if (block\_duration.count() > meaningful\_block\_time)

{

blocked\_reads++;

}

std::cout << "Читатель " << id << " читает данные.\n" << std::flush;

std::this\_thread::sleep\_for(std::chrono::milliseconds(read\_dist(gen)));

successful\_reads++;

WaitForSingleObject(counter\_mutex, INFINITE);

if (--readers\_count == 0)

{

ReleaseSemaphore(resource\_semaphore, 1, NULL);

SetEvent(writer\_cv);

}

ReleaseMutex(counter\_mutex);

}

}

void writer(int id)

{

while (true)

{

std::this\_thread::sleep\_for(std::chrono::milliseconds(delay\_dist(gen)));

auto start\_time = std::chrono::high\_resolution\_clock::now();

WaitForSingleObject(counter\_mutex, INFINITE);

writers\_count++;

ReleaseMutex(counter\_mutex);

while (readers\_count > 0)

{

WaitForSingleObject(writer\_cv, INFINITE);

ResetEvent(writer\_cv);

}

WaitForSingleObject(resource\_semaphore, INFINITE);

auto end\_time = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> block\_duration = end\_time - start\_time;

if (block\_duration.count() > meaningful\_block\_time)

{

blocked\_writes++;

}

std::cout << "Писатель " << id << " записывает данные.\n" << std::flush;

std::this\_thread::sleep\_for(std::chrono::milliseconds(write\_dist(gen)));

successful\_writes++;

WaitForSingleObject(counter\_mutex, INFINITE);

writers\_count--;

if (writers\_count > 0 && readers\_count\_waiting < ((num\_readers - 1) < 1 ? 1 : (num\_readers - 1)))

{

SetEvent(writer\_cv);

}

else

{

SetEvent(reader\_cv);

}

ReleaseMutex(counter\_mutex);

ReleaseSemaphore(resource\_semaphore, 1, NULL);

}

}

void print\_statistics()

{

while (true)

{

std::this\_thread::sleep\_for(std::chrono::seconds(5));

std::cout << "\n--- Статистика ------\n";

std::cout << "Успешных чтений: " << successful\_reads.load() << "\n";

std::cout << "Блокированных чтений: " << blocked\_reads.load() << "\n";

std::cout << "Успешных записей: " << successful\_writes.load() << "\n";

std::cout << "Блокированных записей: " << blocked\_writes.load() << "\n";

std::cout << "---------------------\n";

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

resource\_semaphore = CreateSemaphore(NULL, 1, 1, NULL);

counter\_mutex = CreateMutex(NULL, FALSE, NULL);

reader\_cv = CreateEvent(NULL, TRUE, TRUE, NULL);

writer\_cv = CreateEvent(NULL, TRUE, TRUE, NULL);

if (resource\_semaphore == NULL || counter\_mutex == NULL || reader\_cv == NULL || writer\_cv == NULL)

{

std::cerr << "Не удалось создать синхронизаторы.\n";

return 1;

}

ResetEvent(writer\_cv);

std::vector<std::thread> readers;

for (int i = 0; i < num\_readers; i++)

{

readers.emplace\_back(reader, i + 1);

}

std::vector<std::thread> writers;

for (int i = 0; i < num\_writers; i++)

{

writers.emplace\_back(writer, i + 1);

}

std::thread stats\_thread(print\_statistics);

for (auto& t : readers)

{

t.join();

}

for (auto& t : writers)

{

t.join();

}

stats\_thread.join();

CloseHandle(reader\_cv);

CloseHandle(writer\_cv);

CloseHandle(resource\_semaphore);

CloseHandle(counter\_mutex);

return 0;

}